CLEMSON
School of COMPUTING

Amortized analysis

Amortized analysis

- In an amortized analysis, we estimate the time required to perform a sequence of operations.

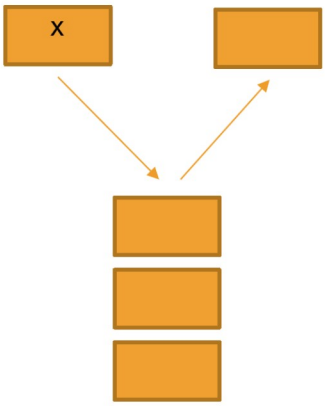
- With amortized analysis, we have the opportunity to show that the average cost of an operation is small, even if the cost of such operation looks big (according to asymptotic notation)

Amortized analysis

Two techniques

- Accounting method
  - Faster to use, intuitive. Better suited for simple operations
- Potential method
  - More formal and structured. Better suited for involved algorithms

First example



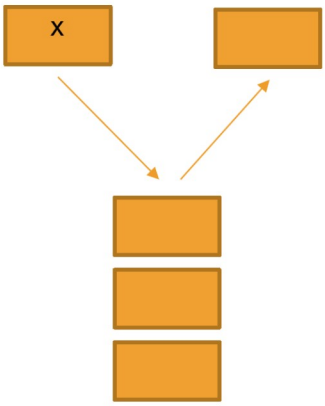
Consider a stack data structure

- Push (S, x) – O(c) ≃ θ(1)
  - Input:
    - S stack
    - x element to insert on top of S
- Pop (S) – O(c) ≃ θ(1)
  - Input:
    - S stack
  - Output
    - x element removed from the top of S

c = constant

Stack: Last in, first out

First example



Suppose you want to run n operations of type POP and PUSH.

What will be the total cost?

Pick the maximum cost among the operations you may run.

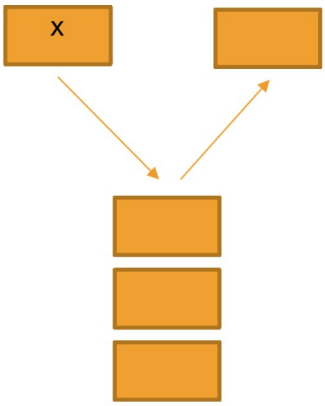
O(c)

Multiply for the times you are running these operations

O(n)

First example

Suppose we add a new operation to our stack



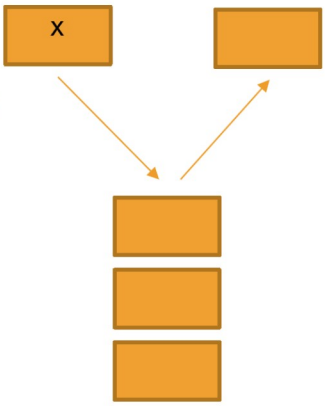

```
function MULTIPOP (S, k){
        input
        - S stack data structure
        - k, number of elements to remove from S
        while (not S.empty() and k>0){
                POP(S)
                k = k-1
        }
}
```

What is the cost of the new operation?

O(k)

k = parameter integer, not constant

First example



Suppose you want to run n operations of type POP, PUSH and MULTIPOP.

What will be the total cost?

Pick the maximum cost among the operations you may run.

$O(k)$

Multiply for the times you are running these operations

$O(kn) = O(n^2)$

Accounting Method

- In the ***accounting method*** of amortized analysis, we assign differing charges to different operations, with some operations charged more or less than they actually cost. We call the amount we charge an operation its ***amortized cost***.

- We assign the difference to specific objects in the data structure as ***credit***.

- We must ensure that the total amortized cost of a sequence of operations provides an upper bound on the total actual cost of the sequence.

$$\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$$

Amortized cost

Real cost

Accounting method

	REAL COST	SCHEMA	
POP	1	0	O(c)
PUSH	1	2	O(c)
MULTIPOP	k	0	O(c)

(number of pops)

push gets 2 credits because pop & multipop rely on something bc pushed

$$\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$$

schema Real cost

Pick the maximum cost among the operations you may run.

O(c)

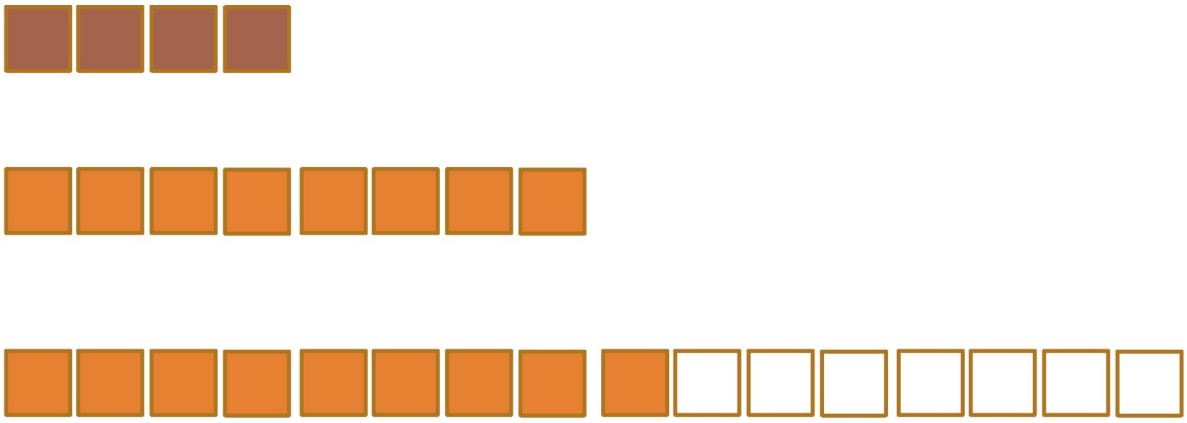
Multiply for the times you are running these operations

O(n)

Schema works as worst-case for all operations.

Dynamic table

- We implement a dynamic array that can be resized every time there is no space for a new element



Dynamic table - algorithm

- Simple insert – $O(c)$
  - When the array has an empty spot
- Complex insert – $O(k)$ when no empty, need to copy K times
  - Resize + insert
  - When the array is full

How does it take to run n inserts?

$O(k)*n = O(n)*n = O(n^2)$

Accounting method

	Real cost operation	Schema	
Insert	1	3	
Resize	k	0	

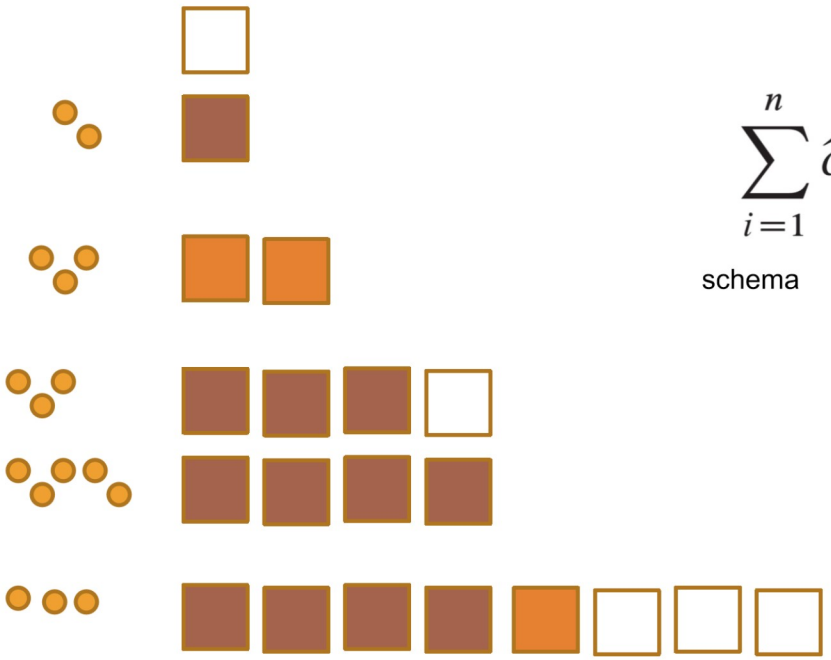
Pick the maximum cost among the operations you may run.

O(c)

Multiply for the times you are running these operations

O(n)

Accounting method



$$\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$$

schema Real cost

What's next?

- Book chapters
  - Chapter 1.4
- In class activities
  - Q&A session
- Next class
  - Amortized analysis with the potential method